JAVA Basics

ver 1.0

(very incomplete and not always completely accurate)

Notation:

… several things, often repetition of the items before and after it

[…] optional construct, except for its use with arrays

… | … | … alternatives, except for its use for the 'or' operation in Boolean expressions

italics font a description of what should appear in a location

Class and interface:

[import *mainPackage.subPackage.ClassName*; …]
public class *Name* [extends *Name2*] [implements *Name3*, … , *Name4*]
{
    constructors, fields, and methods in any order
}

public interface *Name* [extends *Name3*, …, *Name4*]
{
    public constants, and public abstract methods classes in any order
}

Note that each class/interface is in its own file that has the same name as the class/interface and extension .java

Comments:

/* multi-line comment */ // comment for the rest of the line

Variable declarations:

int i, j = 3, k; //other types: byte, short, long, char
float x, y = 4.3f; // need the "f" to obtain a float literal, otherwise double
double d, e = 4.3, f = 5e3;
boolean a, b = true, c = false;
final double MY_PI = 3.14159265; // constant
String s, t = null, u = "Example";
MyType f, g = null, h = new *MyType*(…);

Constructor and method:

public *ClassName* (*Type name, Type name, … Type name*) // need the parenthesis even if no arguments
{
    declarations, and statements
}

public [void | *Type*] *methodName* (*Type name, Type name, … Type name*) [throws *exception1*, … *exception2*]
// need the parenthesis even if no arguments
{
    declarations, and statements
}

Expressions:
Arithmetic operators: + - * /
Note the division of 2 integers results in an integer value obtained by truncating any decimal digits
% remainder (fractional part of a division)
++ unary operator to increment
-- unary operator to decrement
Logical operators: && (and), || (or), ! (not)
Relational operators: <, <=, >, >=, = = (no space between them), != , equals( ), compareTo( )
// for object comparison, especially Strings, usually use equals( ) or compareTo( )
(*NewType*) *expression* // cast the expression to type NewType; only permitted in certain situations
// Any numeric value can be cast to any numeric type, but accuracy might be lost.
// The cast is necessary if accuracy might be lost, eg. long to float.
this // the object within which execution is currently taking place
accessorName(*arg1*, ... *agr2*) // for a routine invocation, need the parenthesis even if no arguments

Statement:
{ ... } // used to group together a sequence of statements to form one statement

variable = *expression*;

modifierName(*arg1*, ... *arg2*); // need the parenthesis even if no arguments

if (*booleanCondition*)
statement1 // use a block for multiple statements
[else
statement2] // use a block for multiple statements

while (*booleanCondition*)
statement // use a block for multiple statements

for (*declarationWithInitialization* | *assignment*; *booleanCondition*; *assignment* | *increment* | *decrement*)
statement // use a block for multiple statements

return *expression* ; throw *exceptionExpression* ;

Arrays: // Note that arrays are reference types, and hence are descendants of the Object class
Type[ ] *myArray*; // declaration
myArray = new *Type*[*length*]; // creation
myArray.length /* expression that yields the length used to create the array
Note that there are no parenthesis for length */
myArray[*index*] = *value*; // Note that the valid index range is 0 to length-1

value = *myArray*[*index*];
myArray = { *value1*, *value2*, ... *valueLast* }; // array literal
Type[ ] [ ] *twoDArray*; // 2-D array

Strings:
myString = "some " + "characters";
myString.length() // Number of characters in the string; note parentheses for String length
myString.equals(yourString) or myString.compareTo(yourString) // don't use = = or !=
// The contents of a String cannot be changed. Use StringBuffer or StringBuilder if it must be changed.

Object: some methods of the Object class are toString( ), equals( ), hashCode( )

Console input/output:
System.out.println("Enter the value for x ");
Scanner *consoleIn* = new Scanner(System.in);
x = *consoleIn*.nextInt( ); // other methods: nextDouble( ), nextLine( ), next( ) //word
System.out.println("The value of x is " + x);